Achieving a high throughput and a low latency through a modular packet scheduler

Maurizio Casoni and Carlo Augusto Grazia*

Department of Engineering Enzo Ferrari,
University of Modena and Reggio Emilia,
via Pietro Vivarelli 10, 41125 Modena, Italy
Email: maurizio.casoni@unimore.it
Email: carloaugusto.grazia@unimore.it
*Corresponding author

Paolo Valente

Department of Physics, Computer Science and Mathematics,
University of Modena and Reggio Emilia,
via Giuseppe Campi 213/a, 41125 Modena, Italy
Email: paolo.valente@unimore.it

Abstract: Providing QoS guarantees in a wireless environment is a challenging task because of the idiosyncrasies of the wireless media. State-of-the-art solutions for QoS provisioning over wireless links are based on cross-layering packet schedulers that deal both with the QoS guarantees and the wireless link issues. Unfortunately, such an approach is not flexible and requires technology-dependent solutions. To address these issues, we present a modular architecture which permits the use of existing high-performance packet schedulers for wired links over generic wireless technologies, as they are, and at the same time allows the flexibility to adapt to different channel conditions. We validate the effectiveness of our modular solution through a formal analysis. We also present high-throughput twin fair scheduler (HFS), a novel packet scheduler based on the modular architecture. HFS has constant execution time, accurate fairness, and low latency.

Keywords: cross-layering; latency; QoS; scheduling; throughput.


Biographical notes: Maurizio Casoni is an Associate Professor of Telecommunications in the Department of Engineering ‘Enzo Ferrari’ (DIEF) at University of Modena and Reggio Emilia (UNIMORE), Italy. He received his MS with honours and his PhD in Electrical Engineering from University of Bologna, Italy, in 1991 and 1995, respectively. In 1995, he was with the Computer Science Department at Washington University in St. Louis, MO, as a Research Fellow. He was responsible at UNIMORE for the EU FP7 Projects E-SPONDER and PPDR-TC.
1 Introduction

Modern wireless technologies (e.g., 3G, 4G, WiFi) steer nowadays the principles behind the design and implementation of wireless networks. State-of-the-art wireless technologies aim to support increasingly higher data rates for applications such as video streaming, web browsing and file sharing, in both stationary and nomadic/mobile scenarios. Moreover given the growing spread of smartphones and tablets, an increasing number of users access to wireless networks everyday.

This trend puts several limits on how network and/or service providers can effectively supply adequate quality of service (QoS) guarantees to their users. The majority of current wireless systems directly provides some QoS capabilities (e.g., traffic differentiation and traffic prioritisation). To this respect, one of the most important network sub-systems involved in the provision of QoS is the packet scheduler, which properly sets the order in which packets are sent over a given interface, both in the uplink and downlink directions. As showed in Sabrina (2010), Lee et al. (2010) and Formyduval and Thuente (2013), providing high QoS guarantees in wireless systems through a smart packet scheduler represents a challenging topic.

State-of-the-art solutions to concurrently provide QoS guarantees and high throughput are based on cross-layering techniques; i.e., where the scheduling decisions are also made using channel state information coming from the MAC layer (Ng et al., 1998; Lu et al., 1999; Yi et al., 2000; Iera et al., 2007). For example, per-destination channel conditions may be considered when choosing which flow to serve, in order to avoid transmission failures. In the aforementioned proposals, just one integrated scheduler takes all the scheduling decisions, based on the detected issues and on the desired QoS.

Unfortunately, integrated solution entails a few drawbacks. A high-quality scheduler for reliable links (Bennet and Zhang, 1997; Valente, 2007; Shreedhar and Varghese, 1995; Vasiliadis et al., 2012; Ramabhadran and Pasquale, 2006; Yuan and Duan, 2009;
van Leeuwaarden et al., 2006) cannot be used and converted into a cross-layer scheduler, not without modifying it. Even after the necessary modifications are done, the guarantees provided by the scheduler are likely to change and would therefore need to be recomputed. Finally, if the medium access protocol or the channel technology changes, or if we want to use new techniques for achieving an even higher throughput or saving energy, then the scheduler is likely to not fit the new technology or requirement. Hence, it may need to be modified again. Especially if we want to use the same scheduler on heterogeneous wireless technologies, we may need to define a different version of it for each technology, which leads to a set of solutions each tailored for a specific technology. In this work, we analyse a packet-scheduling architecture originally proposed in Casoni et al. (2013).

The architecture focuses on local packet schedulers, i.e., those executed inside wireless nodes, and decides (only) the order in which packets are transmitted over the nodes outgoing links. Such architecture preserves both effectiveness and flexibility, and permits to reuse existing schedulers without any modification; the scheduler picks the next packet to transmit according to its QoS policy, but delivers it to a middle layer instead of delivering it to the MAC layer. This middle layer then deals with the issues of the wireless medium and reorders packet transmissions if needed. With this architecture it is easy to cherry pick from the literature and combine the best solutions in terms of service guarantees, computational cost, power consumption and throughput boosting. As for the last two goals, we note that this architecture allows a system to profit from cross-layering while still preserving flexibility.

The contribution of this paper is threefold:

1. We complete the modular architecture by defining a mechanism that moves the packets from the QoS layer to the middle layer. This mechanism is implemented in a software module called packet prefetcher and it is based on a shared buffer of size $Q$ with virtual queueing.

2. We provide a mathematical formulation in order to compute the overall guarantees of the entire architecture as a function of QoS layer and middle layer. In this way, we formally define the guarantees perturbations caused by the introduction of the second scheduler.

3. We leverage the key property of the modular architecture (i.e., to allow an easy investigation of several schedulers’ combinations) by defining a new scheduler with accurate performance and low energy consumption, called high-throughput twin fair scheduler (HFS). Performances of HFS are then validated through extensive simulations.

HFS provides two contributions to energy reduction. Firstly, by increasing the throughput, it increases the number of packets that are successfully transmitted per fixed energy consumed (the number of retransmissions lowers); secondly, according to our experiments, the time and the energy needed to execute HFS for each packet to enqueue/dequeue are close to those of a deficit round robin (DRR).

The rest of the paper is organised as follows. Section 2 describes the related works. In Section 3 we describe the modular architecture and in Section 4 we compute its overall guarantees. Then in Section 5 we show the testbed that has been used to deploy the architecture and define HFS. In Section 6, we present the new packet scheduler for wireless links. In Section 7, we validate HFS by comparing it with the best
Achieving a high throughput and a low latency

high-performance schedulers for wired links and with the best integrated scheduler. Finally, in Section 8 we highlight our conclusions.

2 Related work

Cross-layering techniques have been proposed initially in Ng et al. (1998) to adapt the simple packet fair queueing algorithms to move from wired to wireless networks. The idea has been to include location-dependent channel errors in the scheduling decision, collecting these channel errors and passing the channel information to the higher TCP-IP stack layers. Continuing with this approach, other works such as Lu et al. (1999), Yi et al. (2000) and Iera et al. (2007) have used the same technique to improve and adapt existing schedulers to work properly on wireless networks. Unfortunately, all of these works are tailored on a specific packet scheduler for a specific wireless link, without the possibility to extend these solutions to a general purpose scenario.

The same approach has been used to create novel packet schedulers for specific wireless links. This is the case of Cohen and Grebla (2014) and Grebla et al. (2015) for the LTE technology, (Cicconetti et al., 2014; Ke et al., 2014) for WiMAX technology, and (Szymanski and Gilbert, 2008) for cluster systems. In these works, the synergy between the selected packet scheduler and the medium technology is even higher, considering characteristics bounded and tailored for the specific problem under exam. Continuing, a lack in flexibility is the key drawback for these solutions.

The necessity to guarantee tight QoS requirements even over wireless links have produced also many research contributions on related topics. An example is the bandwidth scheduling over dedicated channel proposed in Zuo et al. (2015) or, more related to our specific approach, the use of cross-layering techniques to boost the QoS for a routing protocol in Sondi et al. (2014). This is also the case of Al-Oudat and Manimaran (2012) where QoS requirements have been studied together with the security aspect by using of graph theory.

Considering the idea to combine two different packet-level algorithms (e.g., in our proposal two packet-schedulers are combined), a new approach is emerging in the literature. This approach consists in combining a packet scheduler with an active queue management (AQM) technique to solve both the QoS and the congestion issues. This solution have started to be proposed after the definition of the bufferbloat problem (Gettys and Nichols, 2012). The first example of this multi-purpose solution has been FQ-Codel (Grigorescu et al., 2015) where the DRR scheduler has been combined with the CoDel AQM and, continuing, many AQM are under study in the IETF community (and not only) for upgrading to this combination. This is the case of PIE (Pan et al., 2013), PINK (Grazia et al., 2015) and QRM (Casoni et al., 2015b) as an example.

It is important to note that the solution that we are proposing in this paper is able to accommodate these last outcomes of the research community. This is discussed in the next section where the modular architecture is described.

3 Architecture

The architecture is composed by two layers: a QoS provisioning layer, hereafter called just QoS layer for short, and a MAC scheduling and abstraction layer, hereafter
abbreviated as MAC-SAL. The purpose of the former is to put together, conceptually, the two most important components for providing service guarantees over a transmission link: the packet classifier and the packet scheduler. Packets are passed from the QoS layer to the MAC-SAL by a packet prefetcher, described in detail in Subsection 3.3. The architecture model for a system containing only one outgoing link is shown in Figure 1, while a generalisation of this architecture for a multi-link system is discussed in Casoni et al. (2013).

**Figure 1** Modular architecture for providing QoS over a wireless link (see online version for colours)

### 3.1 QoS layer

The QoS layer is shown in the top box of Figure 1. We added the prefix QoS to the name of both the classifier and the scheduler in this layer to highlight their goal and to distinguish them from the corresponding components in the MAC-SAL (that are in turn...
Achieving a high throughput and a low latency

described in the next subsection). As for the classifier, it divides packets into \( N \) separate flows, so that individual bandwidth and delay guarantees can be provided to each flow. Finally, the purpose of the scheduler is to enforce a scheduling policy that does provide the desired guarantees.

To complete the description of the QoS layer, let us consider the path followed by a packet passed to this layer by invoking the send() interface function (top left corner in Figure 1). The packet first enters the QoS classifier, which determines the QoS flow it belongs to; then it is inserted into the queue associated to the flow. A possible additional intra-flow scheduler or AQM can be added, in this way, this architecture is able to accommodate the novel researches regarding the combination of packet scheduling with AQM techniques. The next packet to serve can be requested to the QoS layer by invoking the get_next_pkt() function. By calling this function, the next packet is chosen among the head-of-line packets of the QoS flows. This packet is then removed from the queue and passed to the entity that invoked the function.

3.2 MAC scheduling and abstraction layer

The MAC-SAL has the same structure of the QoS layer. What changes is the purpose of the components, and of the layer as a whole. In particular, on one side the MAC-SAL interacts with the underlying MAC layer and deals with the details of the wireless technology at hand (the possible information received by the MAC-SAL from the MAC layer are reported on the right of the box representing the MAC-SAL in Figure 1). On the other side, the MAC-SAL exports an interface made by two functions: MAC-SAL-send() and link_ready(), where the latter informs when the abstract link is ready.

The MAC-SAL may serve various important purposes. Firstly, if required by the application at hand, it may make sure that packets get eventually transmitted with success. Secondly, it may implement algorithms for the maximisation of link throughput; to achieve this, the MAC-SAL must classify packets according to their chances of successful transmission, and must be able to change the order in which packets are sent to the MAC layer. In the end, although with a different goal, the MAC-SAL must, in general, accomplish the same main sub-tasks as the QoS layer: divide packets into distinct flows, store the packets of each flow in a distinct queue and schedule the head-of-line packets of the flows according to the desired policy.

3.3 Packet prefetcher

The higher is the number of non-empty MAC-SAL queues, i.e., flows, the higher is the number of head packets among which the MAC-SAL scheduler can choose the next packet to transmit. Hence, the higher is the probability that the MAC-SAL scheduler can pick good packets with respect to the goals it wants to achieve. For example, suppose that the goal of the MAC-SAL scheduler is to keep a high throughput and that some MAC-SAL queues contain packets for destinations with bad channel conditions. If many/few other queues are not empty, then the probability for the scheduler to have at its disposal better packets to transmit is high/low. In the end, to maximise the effectiveness of the MAC-SAL, its queues should be always kept as full as possible. This is exactly the purpose of the packet prefetcher depicted in the middle of Figure 1.

The behaviour of the prefetcher depends on a parameter \( Q \), which is the MAC-SAL input buffer size, measured in number of bytes. We say that the prefetcher prefetches a
packet if it invokes the function get_next_pkt() to get the next packet, say pkt, from the QoS layer and then invokes MAC-SAL-send(pkt) to insert the packet in the MAC-SAL. We assume that a prefetched packet is never dropped by the MAC-SAL. Finally, we denote as $S(t)$ the sum of the sizes of the packets queued in the MAC-SAL at time $t$.

The prefetcher operates on each of the following two events: when a new packet arrives in the QoS layer and when a new packet is dequeued from the MAC-SAL. On the former, let $pkt$ be the new arriving packet. If $S(t) < Q$ when $pkt$ arrives, then the packet prefetcher immediately prefetches $pkt$. On the latter, if $S(t)$ becomes lower than $Q$ when the packet is dequeued from the MAC-SAL, then the packet prefetcher starts prefetching new packets until $S(t) \geq Q$. In other words, the combination of the set of queues in the MAC-SAL and of the packet prefetcher implements a shared buffer with virtual queuing, a device in which the memory used to store the packets is shared and the queues are only virtually separated.

Finally, it is worthwhile noting that this scheme is much more effective than an approach with distinct queues and with the same total memory. As for the latter, suppose for example that all the queues have the same length, equal to $Q/M$ plus one maximum packet size $L$. If all the packets prefetched in a given time interval are destined to the same MAC-SAL queue, then the queue becomes full only after $Q/M + L$ bytes have been prefetched. If the next packet to prefetch is again destined to the same queue, then the prefetcher must block it. In the shared-buffer scheme, instead, the prefetcher must block a packet only after $Q + L$ bytes have been prefetched, and this increases the probability that more queues are not empty.

4 Service guarantees

In this section we define first a simple byte-delay parameter to model the MAC-SAL. From this parameter we immediately derive the per-packet delay that the MAC-SAL introduces. Using this special delay parameter we also compute the amount of service guaranteed to each QoS flow over any time interval. Finally, from the latter service property, we compute the bandwidth guaranteed to each flow over any time interval. This last guarantee is not particularly meaningful for short time intervals, during which the bandwidth received by a flow is likely to be null or extremely low. In fact, packet delays can cause a flow to receive little or no service during a short time interval. This guarantee is instead useful for computing the long-term bandwidth received by a flow as the size of the time interval grows. For space limitations, in this paper we do not compute other service metrics, as, e.g., worst-case fair index (Bennet and Zhang, 1997) or relative fairness (Golestani, 1994) (they can however be easily derived from the service properties reported in this section).

We compute all the following quantities assuming that no packet is dropped by the MAC-SAL because some MAC queue is full. In practice, we assume that the offered load for each MAC queue is, on average, not higher than the rate at which the MAC scheduler guarantees that the queue is emptied. There may be temporary overloads, but the maximum size of the queue is never exceeded. As for packet dropping, consider the overall scheduling policy implemented by the combination of the QoS layer and the MAC-SAL. If this policy leads to packet dropping for memory constraints, then the same problem would occur with an integrated scheduler enforcing the same policy.
Achieving a high throughput and a low latency

We can now introduce the special quantity from which we derive all the service guarantees reported in this section. To this purpose we consider that the MAC-SAL influences service guarantees mainly because it may reorder packets, and hence may not respect the service order suggested by the QoS scheduler. We take this fact into account by modelling the abstract link implemented by the MAC-SAL as a special link characterised by a byte delay due to packet reordering. Given any packet $p$ just queued in the MAC-SAL, we define the byte delay of the abstract link as the maximum possible value of the sum of the sizes of the packets that are queued after $p$ but are transmitted over the outgoing link before $p$. We assume that the byte delay experienced by a packet is a function of the flow it belongs to, and we denote as $\Delta_i(t_1, t_2)$ the byte delay experienced by the packets of the $i^{th}$ flow during a generic time interval $[t_1, t_2]$. Of course, for all flows $\Delta_i(t_1, t_2) = 0$ if packets are transmitted in the same order as established by the QoS scheduler.

Consider a packet $p$ entering the MAC-SAL at time $t_1$ and exiting from the MAC-SAL at time $t_2$. From the byte delay we can immediately derive the component $D_i(t_1, t_2)$ of the worst-case delay of $p$ due to the fact that the MAC-SAL reorders packets. If we denote as $B(t_1, t_2)$ the minimum bandwidth of the outgoing link during $[t_1, t_2]$, then

$$D_i(t_1, t_2) \leq \frac{\Delta_i(t_1, t_2)}{B(t_1, t_2)}.$$  \hspace{1cm} (1)

We compute now the amount of service guaranteed to the $i^{th}$ QoS flow in a generic time interval $[t_1, t_2]$ during which the $i^{th}$ QoS flow is continuously backlogged, i.e., has pending packets either in the QoS-scheduler or in the MAC-scheduler. We express this guarantee as a function of two further variables. The first is the maximum possible value $Q$ for the sum of the sizes of the packets that can be present in the MAC-SAL at the same time. The second is the minimum fraction of the link bandwidth guaranteed by the QoS scheduler to the $i^{th}$ flow during $[t_1, t_2]$ if all the packets queued in the MAC-SAL are served in the same order as they are dequeued from the QoS scheduler. We denote as $\alpha_i^{QoS}(t_1, t_2, Q)$ this quantity. We assume that $\alpha_i^{QoS}(t_1, t_2, Q)$ is a function of also $Q$ for two important reasons. The first is that queueing packets in the MAC-SAL before serving them introduces a delay between when a packet is dequeued from the scheduler and when the packet starts to be transmitted. This delay of course influences the guarantees provided to a flow. The second reason is that the sum of the sizes of the packets dequeued, but not yet transmitted, negatively affects the time-stamping rules and hence the service guarantees of many schedulers. The following theorem shows that the service lost by the $i^{th}$ flow because of the byte delay is, in the worst-case, equal exactly to $\Delta_i(t_1, t_2)$.

**Theorem 1:** Let $W_{i}^{QoS}(t_1, t_2, Q)$ be the minimum number of bytes of the $i^{th}$ flow that the QoS scheduler would guarantee to be transmitted during $[t_1, t_2]$ if the link bandwidth was constant and equal to $B(t_1, t_2)$ and if $\Delta_i(t_1, t_2) = 0$, i.e., if all the packets queued in the MAC-SAL were served in the same order as they are dequeued from the QoS scheduler. Let $W_i(t_1, t_2, Q)$ be the minimum number of bytes of the $i^{th}$ flow that are actually guaranteed to be transmitted if only the first hypothesis holds, i.e., if the link bandwidth is constant and equal to $B(t_1, t_2)$. The following inequality holds:

$$W_i(t_1, t_2, Q) \geq W_{i}^{QoS}(t_1, t_2, Q) - \Delta_i(t_1, t_2).$$  \hspace{1cm} (2)
**Proof:** consider the sequence of packets of the \(i\)th flow that the QoS scheduler would guarantee to be transmitted during \([t_1, t_2]\) if the link bandwidth was constant and equal to \(B(t_1, t_2)\) and if \(\Delta(t_1, t_2) = 0\) held. A possible sequence of such packets is shown in Figure 2. Each packet is depicted as a rectangle with the following property: the projection onto the \(x\) axis of its left/right side is equal to the start/finish time of the packet it represents. Especially, in the figure the packet \(p_i/p_{k}\) starts/ends to be transmitted before/after time \(t_1/t_2\). According to the figure, \(W_{i}^{QoS}(t_1, t_2)\) is equal to the sum of the sizes of the portions of the packets \(p_1\) and \(p_{k}\) served after \(t_1\) and before \(t_2\), plus the sizes of the packets \(p_2\) through \(p_{k-1}\).

Figure 2  Illustration of how the byte delay reduces the service received by a flow (proof of Theorem 1)

To prove the thesis, we consider, with the help of the figure, what packets and portions of packets may not be served any more during \([t_1, t_2]\) if \(\Delta(t_1, t_2) > 0\). Let \(\Delta T_i\) be the time needed to transmit \(\Delta_i(t_1, t_2)\) bytes at speed \(B(t_1, t_2)\), i.e., \(\Delta T_i = \frac{\Delta_i(t_1, t_2)}{B(t_1, t_2)}\) (recall that we assume that the link bandwidth is constant and equal to \(B(t_1, t_2)\) during \([t_1, t_2]\)). In the worst-case, the effect of the byte delay \(\Delta_i(t_1, t_2)\) introduced by the MAC-SAL is letting the transmission of all the packets in Figure 2 start \(\Delta T_i\) time units later. It follows that the packets and the portion of packets that, if \(\Delta_i(t_1, t_2) = 0\), would start to be transmitted too close to \(t_2\), more precisely not before time \(t_2 – \Delta T_i\), risk to not be transmitted at all during \([t_1, t_2]\). Figure 2 helps us visualise this fact. The start time of the portions of the packets \(p_1\) and \(p_{k}\) transmitted during \([t_2 – \Delta T_i, t_2]\), as well as of the whole packets \(p_1\) through \(p_{k-1}\) may be delayed enough to let none of these portions and full packets be served during \([t_1, t_2]\). In the end, \(W_i(t_1, t_2, Q)\) is, in the worst case, equal to the sum of the sizes of only the portions and the whole packets served during \([t_1, t_2 – \Delta T_i]\) in case \(\Delta_i(t_1, t_2) = 0\) (i.e., in Figure 2, of the portions of the packets \(p_1\) and \(p_{j}\) served during \([t_1, t_2 – \Delta T_i]\)) and of the packets \(p_2\) through \(p_{k-1}\).

To express what we have stated so far with a formula, we denote as \(W_i^{QoS}(t_1, t_2, \Delta T_i, Q)\) and \(W_i^{QoS}(t_2, -\Delta T_i, t_2, Q)\) the sums of the sizes of the full packets and the portions of packets transmitted, if \(\Delta_i(t_1, t_2) = 0\), during \([t_1, t_2 – \Delta T_i]\) and \([t_2 – \Delta T_i, t_2]\), respectively. It follows that:

\[
W_i(t_1, t_2, Q) \geq W_i^{QoS}(t_1, t_2 - \Delta T_i, Q) = W_i^{QoS}(t_1, t_2, Q) - W_i^{QoS}(t_2 - \Delta T_i, t_2, Q).
\] (3)

The get the thesis from the above inequality we consider that \(W_i^{QoS}(t_2, -\Delta T_i, t_2, Q)\) is at most equal to the number of bytes that can be transmitted by the link during \([t_2 – \Delta T_i, t_2]\).

Since we assume that the link works at constant speed \(B(t_1, t_2)\), this number of bytes is equal to \(B(t_1, t_2) \cdot \Delta T_i = \Delta_i(t_1, t_2)\).

Finally, in the following theorem we report the bandwidth guaranteed to each QoS flow. To write the theorem, we define a last symbol, \(B_i(t_1, t_2, Q)\), by which we denote the
Achieving a high throughput and a low latency

minimum bandwidth guaranteed by the system to the \( i \)th QoS flow during \([t_1, t_2]\) in the worst-case, i.e., if the link bandwidth is constant and equal to the minimum bandwidth \( B(t_1, t_2) \).

**Theorem 2:** given any time interval \([t_1, t_2]\) during which the \( i \)th QoS flow is continuously backlogged, we have

\[
B_i(t_1, t_2, Q) \cdot \alpha^{QoS}_i(t_1, t_2, Q) \cdot B(t_1, t_2) - \frac{\Delta_i(t_1, t_2)}{t_2 - t_1}
\]

(4)

**Proof:** by the definitions of \( \alpha^{QoS}_i(t_1, t_2, Q) \) and \( B(t_1, t_2, Q) \), the following equalities hold for the quantities defined in Theorem 1: \( W^{QoS}_i(t_1, t_2, Q) = \alpha^{QoS}_i(t_1, t_2, Q) \cdot B(t_1, t_2) \) and \( W_i(t_1, t_2) = B(t_1, t_2, Q) \cdot (t_2 - t_1) \). This allows us to rewrite (2) as follows:

\[
B_i(t_1, t_2, Q) \cdot (t_2 - t_1) \geq \alpha^{QoS}_i(t_1, t_2, Q) \cdot B(t_1, t_2) \cdot (t_2 - t_1) - \Delta_i(t_1, t_2)
\]

(5)

We get the thesis by dividing both sides by \( t_2 - t_1 \). □

In the next subsection we make some observations on the above service guarantees and on their relationship with the characteristics of an actual system.

4.1 Discussion

First, to use (1), (2) and (4), it must be possible to model the components of the architecture in terms of the variables that appear in those inequalities. In this respect, we note that these variables express the minimal guarantees that must be provided by these components for the whole system to provide any guarantee. In fact, if the link cannot guarantee a minimum bandwidth, or the scheduler cannot guarantee a minimum fraction of that bandwidth, or, finally, if the MAC-SAL cannot guarantee that a queued packet gets eventually transmitted, then, of course, no overall guarantee can be provided at all.

Besides, the bound (4) shows that the MAC-SAL affects per-flow bandwidths in both an explicit and an implicit way. The explicit way is through \( \Delta_i(t_1, t_2) \). If this quantity grows less than linearly with \( t_2 - t_1 \), then, in the long term, the fraction of the bandwidth guaranteed to the \( i \)th flow tends to the ideal fraction that would have been guaranteed by the QoS scheduler alone. In contrast, if \( \Delta_i(t_1, t_2) \) grows at least linearly with \( t_2 - t_1 \), then the \( i \)th flows may get, in both the short and the long term, a lower fraction of the bandwidth than desired. The implicit influence on the bandwidth guaranteed to each flow stems instead from the fact that the MAC-SAL influences the value of \( B(t_1, t_2) \) itself in (4).

To lay down some guidelines for guaranteeing a low \( \Delta \), consider a sequence of packets to send to a next hop characterised by a bad signal or by other transmission problems. These packets will unavoidably experience high delays. If some of them carry more time-sensitive information than others, it may be important to grant to the former at least a differentiated service. The simplest way to achieve this goal is using separate MAC queues for them and a careful enough scheduling algorithm. Another possibility is dequeuing them before other lower-priority packets inserted in the same queues. This is the main reason why we added in-flow schedulers in the MAC-SAL in Figure 1.
4.2 Guarantees instantiation

What’s happen if we attach the QoS scheduler to the MAC-SAL layer instead of directly to the transmission link? Here we answer, in a sense, to this question by deriving the total perturbation of the service guarantees caused by the MAC-SAL scheduler.

In Rizzo and Valente (2015) it has been proved that the actual guarantees provided by any scheduler depend on the presence and the size of a FIFO transmit queue, such as a buffer ring in a modern NIC. These queues are typically used to drive communication devices and to absorb link-feeding latencies. In more detail, both the packet completion times and the service lag guaranteed by any scheduler happen to always contain an additional variation component equal to, respectively, \( \frac{Q}{R} \) (on a link with a constant rate \( R \)) and \( Q \), where \( Q \) is equal to the size of the transmit queue in bytes (Rizzo and Valente, 2015).

For a timestamp-based scheduler, these components are smaller than expected at a first glance. In fact, they are equal to the minimum possible deviation from the original service that the presence of the queue may cause: the first component is equal to just the time to empty the queue and the second component is equal to just the size of the queue. In contrast, the presence of the queue also affects timestamp computation. In this respect, the simplest scheme for computing timestamps in presence of an intermediate queue before the transmission link is updating flow timestamps on each packet dequeue as if the just dequeued packet was immediately transmitted (Rizzo and Valente, 2015). As a consequence, if a packet arrives, in the worst case, when the queue is full, then the corresponding flow is time stamped as if the system had already transmitted all the packets still in the queue. Despite this large timestamp perturbation, the scheduler does not however suffer from further delay and lag components, apart from the above \( \frac{Q}{R} \) and \( Q \).

We can use this result to easily derive the modification of the service guarantees of the QoS scheduler caused by the presence of the MAC-SAL. To this purpose, we denote as \( Q' \) the maximum number of bytes that can be dequeued from the MAC-SAL after a packet \( p \) is inserted into it, and before \( p \) is dequeued from it. It follows that the queueing delay in the MAC-SAL is at most \( \frac{Q'}{R} \), i.e., it is equal to the queueing delay in a FIFO queue of size \( Q' \). In contrast, the actual buffer size is just \( Q \), which is usually much smaller than \( Q' \). Hence, according to the above arguments, the timestamp perturbation is equal to that caused by a FIFO queue of size \( Q \), and therefore much smaller than that caused by a FIFO queue of size \( Q' \). In the end, we can safely conclude that the additional packet delay variation caused by the MAC-SAL, with respect to the original time guarantees of the QoS scheduler, is not higher than \( \frac{Q'}{R} \), whereas the additional service-lag variation is at most \( Q \).
5 Test environment

In this section we present the experimental setup used to deploy, test and validate HFS and its modular structure. We configure our experiments by using a test environment called TEMPEST (Casoni et al., 2014b), extended to test schedulers over the modular architecture solution showed in Section 3; the modular structure coded in the test environment is depicted in Figure 3. TEMPEST is a novel tool created for evaluating the actual packet schedulers’ performance in several and realistic operational scenarios. With the help of this flexible tool it is possible to measure and compare each scheduler, against any other scheduler, over execution time, simulated throughput and QoS performance. An existing packet scheduler can be easily plugged into this environment, after at most some little interface changes. Inside TEMPEST, the scheduler can then be exercised with the desired sequence of enqueue/dequeue requests, through a controller (see the top of Figure 3) that iteratively switches between two phases: an enqueue phase in which it generates fake packets by picking them from a free list, and a dequeue phase in which it dequeues packets from the scheduler and reinserts them into the free list. The switch occurs according to two configurable max-total-backlog and min-total-backlog thresholds. An overview of the parameters is available in Table 1 together with the default values used in the simulations of this paper, continuing in this section the test environment is described in details with its characteristics.

Figure 3 Modular architecture deployed in TEMPEST
### Table 1  
Configuration parameters: short description and default values used in the simulations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of events</td>
<td>Number of enqueue and dequeue events used to stress the scheduler under exam</td>
<td>10 M</td>
</tr>
<tr>
<td>Packets size</td>
<td>Amount of memory used to store each packet</td>
<td>1,700</td>
</tr>
<tr>
<td>Min pending packets</td>
<td>Minimum amount of packets backlogged in the scheduler</td>
<td>0</td>
</tr>
<tr>
<td>Max pending packets</td>
<td>Maximum amount of packets backlogged in the scheduler</td>
<td>300</td>
</tr>
<tr>
<td>Packet arrival pattern</td>
<td>Packets arrive for each flow in a round robin fashion in two possible way:</td>
<td>Bursty</td>
</tr>
<tr>
<td></td>
<td>smooth: 1 packet per flow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bursty: packets proportional to the flow weight</td>
<td></td>
</tr>
<tr>
<td>QoS flows</td>
<td>Amount of flows coming from the upper layers</td>
<td>100</td>
</tr>
<tr>
<td>QoS flows weight</td>
<td>The higher the flow weight is the higher its priority is</td>
<td>60 flows with weight 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 flows with weight 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 flows with weight 20</td>
</tr>
<tr>
<td>Packet scheduler</td>
<td>Define the algorithm to use different schedulers can be used at QoS layer and MAC layer</td>
<td>FIFO, DRR, WF^2^Q^+^, KPS, QFQ^+^</td>
</tr>
<tr>
<td>Buffer size q in pkts</td>
<td>Amount of packet that can be stored in the MAC buffer</td>
<td>From 1 to 250</td>
</tr>
<tr>
<td>Number of MAC flows</td>
<td>Amount of wireless nodes</td>
<td>20</td>
</tr>
<tr>
<td>Integrated threshold</td>
<td>Packet loss probability to distinguish between good and bad nodes</td>
<td>20%</td>
</tr>
<tr>
<td>Channel bandwidth</td>
<td>Wireless media speed</td>
<td>54 Mb/s</td>
</tr>
<tr>
<td>Packet loss per MAC flow</td>
<td>Distribution of packet loss probability through the wireless nodes</td>
<td>Analogical from 10^0 to 10^-4</td>
</tr>
</tbody>
</table>

### 5.1 Configuration

Each test consisted of 10 M events with a typical balancing of 5 M packet enqueues and 5 M packet dequeues, with the controller configured so as to let flows oscillate between a null backlog and a backlog of ten packets each. Such an enqueue/dequeue pattern happened to be the most demanding one for the schedulers. Packets had a fixed size of about 1,700 bytes, with no cache-line alignment. The payload of the packets was never either read or written. No migration and no packet drop occurred in any run.
5.2 Arrival pattern

The controller switches between enqueue and dequeue mode to control the number of pending packets of each run. The packet arrival pattern block (see Figure 3) runs by default two basic arrival patterns:

- smooth pattern: the controller, in the enqueue phase, iteratively generates one packet for each flow with a lower backlog than the other flows (by filling flows in a round-robin fashion)

- bursty pattern: the controller, in the enqueue phase, iteratively generates a burst of packets for each flow with a lower backlog than the other flows, where the burst has a random size proportional to the flow weight.

In addition to the above rules for switching between the dequeue and enqueue phase, if the backlog of a flow drops below a configurable threshold while the controller in dequeue mode (e.g., one packet), then the controller may switch back, with a configurable probability, to the enqueue phase, filling again the queue of this flow and the one of the others. This additional mechanism is used to increase the randomness of the arrival pattern, and hence make harder for a packet scheduler to guarantee a tight packet delay.

The tests described in the paper have been executed with a bursty packet arrival pattern.

5.3 Wireless scenarios

In this subsection we present two general purpose scenarios, their equivalent specific purpose scenarios have been studied for the EU FP7 project ‘public protection and disaster relief – transformation centre’ (Casoni et al., 2015c, 2015a; Grazia et al., 2014) and are depicted in Figure 4.

Figure 4 Emergency network hierarchical start topology (see online version for colours)
The first consists of an IEEE 802.16 (WiMAX)-based network, where a base-station (BS) is connected to several subscriber-stations (SS), adopting a star-based topology. Since any given SS communicates directly solely with the BS, only the latter needs CSI to adapt its transmissions to the channel conditions. The CSI that the BS can sample can be represented by the received signal strength indicator (RSSI) and by the carrier-to-interference-plus-noise ratio (CINR). For both, mean sample value and sample standard deviation are estimated and can be sent back to the BS, by using proper messages defined by the standard, such as the channel measurements report response (REP-RSP), or by using fast feedback channel quality indicator channels (CQICH) (Zhang, 2007). The equivalent of this general purpose scenario in the FP7 Project is the star topology connection between the Mobile Emergency Operations-Control Center (MEOC) and the first responder chiefs (FRc).

The second scenario is composed by an infrastructure-based IEEE 802.11 (or WiFi) network, which is very similar, conceptually, to that envisaged in the previous paragraph about WiMAX. Consider a specific case, where a set of users, each equipped with a WiFi enabled station (STA), are wireless-connected together using a single deployed AP. As the BS in the previous scenario, only the AP needs CSI to adapt the transmission to the channel conditions. This could be the case for several laboratories and offices located on the same floor of a building. In this case, metrics such as RSSI and MIMO settings could be effectively employed to assess the state of the channel. The equivalent of this general purpose scenario in the FP7 Project is the star topology connection between the FRc and the first responders (FR).

In the previously described scenarios (general and/or specific purpose), several applications and services can be effectively deployed. To name a few, video and voice services, together with web-browsing or database query, represent some of the most-used network applications. TEMPEST is able by default to configure both the described scenarios thanks to the CSI module depicted in Figure 3 which is the core of the wireless simulation part. For the sake of brevity, in this paper we present the simulation results for the second scenario. As a side note, we obtained similar results in the first scenario, so our claiming is relevant also for the second one. The simulation parameters are the following:

- total bandwidth: 54Mb/s
- 20 subscriber stations (SS, or FR in Figure 4)
- 2.7Mb/s per SS
- for each SS, 5 flows:
  a. one with ~1.6 Mb/s reserved for video or VoIP (20 flows with weight 20)
  b. another with ~0.8Mb/s reserved for WEB browsing or direct downloads (20 flows with weight 10)
  c. the other three with ~0.1Mb/s reserved for download/sharing (3 • 20 flows with weight 1).

To simulate the wireless link we used the CSI module of TEMPEST in which different signal to noise ratio (SnR) are mapped in different Packet loss probability (P_{loss}) or, equivalently, different flow bandwidth accordingly to the ns-3 WiFi module. For base stations channel condition, we configured the emergency network scenario in order to
Achieving a high throughput and a low latency

experience a packet loss probability ranging linearly from $10^0$ to $10^{-4}$. These different values are used at the MAC-SAL level to configure the packet scheduler behaviour (it will be properly analysed in Section 6.2) by defining different flow weight distributions through a conversion function. The same conversion can be made also by starting from the SnR input values or, equivalently, by using the different instantaneous flow bandwidth achieving the same weight distributions through a complementary conversion function. Due to its simplicity, in the following we will always talk about $P_{\text{loss}}$ and its related conversion function to calculate the MAC-SAL flow weights.

From a user classification point of view, we define also a threshold to distinguish user in good and bad channel condition. This threshold is placed at 20% of packet loss. Users/flows with less or equal to 20% of packets lost are considered as users/flows in good conditions, while users/flows with more than 20% of packets lost are considered as user/flows in bad conditions. This is because above 20% most applications do not work properly; for instance, both the TCP window and the VoIP controller mechanism do not perform well under the aforementioned condition. Moreover, with this threshold, also dumb nodes are captured in the bad group (Roy et al., 2015; Kar et al., 2015).

5.4 Statistics

To measure and validate schedulers’ performance we implemented different well-known metrics. We used all of these indicators to easily design and test different packet scheduler solutions. Here is a list of the main performance metrics and a relative detailed description.

5.4.1 Throughput

To measure the throughput we simulated the normalised throughput achieved by the schedulers. This parameter is the amount of successfully transmitted packets for each flow divided by the amount of total sent packets. We computed the normalised throughput as

$$\text{thr} = \frac{\sum_i pkts_{\text{sent}_i} (1 - P_{\text{loss}_i})}{\sum_i pkts_{\text{sent}_i}}$$

where $pkts_{\text{sent}_i}$ is the number of packets sent by the flow $i$, $P_{\text{loss}_i}$ is the packet loss probability of the flow $i$ and $pkts_{\text{sent}_i}(1 - P_{\text{loss}_i})$ is the number of successfully transmitted packets by the flow $i$.

5.4.2 Execution time

The actual execution time measurement is a key point of TEMPEST. As a matter of fact, it is possible to measure, due to TEMPEST emulation nature, the real execution time of a packet scheduler by running actual Kernel code in user space. At the end of each run, we measured the total execution time of the run and divided it by the total number of enqueues, or equivalently of dequeues, executed. We obtained therefore the average total
packet-processing time, i.e., the execution time of an enqueue plus a dequeue, inclusive also of the cost of generating and discarding an empty, fixed-size packet;

5.4.3 Energy consumption

According to the models in Bartolini et al. (2011) and Sadri et al. (2011), lower/higher relative execution times imply also lower/higher relative energy consumptions. Therefore, we consider the energy consumption parameter as a direct consequence of the execution time of a scheduler;

5.4.4 Queueing delay

The queueing delay for each flow is measured as the maximum amount of time experienced by any packet inside the flow expressed in number of dequeue events. The delay experienced by a packet \( \text{pkt} \) has been calculated as \( t_{\text{deq}} - t_{\text{enq}} \) where \( t_{\text{enq}} \) is the number of packets dequeued by the system when \( \text{pkt} \) is enqueued, while \( t_{\text{deq}} \) is the number of packets dequeued by the system when \( \text{pkt} \) is dequeued.

5.4.5 Time worst-case fair index (T-WFI)

Another useful QoS guarantees metric is the T-WFI [described in Valente (2014)], which allows to evaluate, in a single value, both fairness and delay: on one hand it shows the fairness, in terms of delay from the worst-case completion time of a packet in a perfectly fair system, while on the other hand it allows to instantly calculate the actual delays incurred by packets depending on the occupation of queues. In a perfectly fair system, the worst-case completion time of a packet is equal to its queue length (including the packet itself) divided by the packet’s flow guaranteed rate. Assuming that the link rate \( R \) is constant, we measured T-WFI\( ^k \) for flow \( k \) as:

\[
T\text{-WFI}^k = \max \left( t_{\text{deq}} - t_{\text{enq}} - \frac{Q^k (t_{\text{enq}})}{\phi^k R} \right)
\]

where \( t_{\text{enq}} \) and \( t_{\text{deq}} \) are the same of the queueing delay parameter, \( Q^k (t_{\text{enq}}) \) is the backlog of flow \( k \) just after the arrival of the packet and \( \phi^k \) is the relative weight of the flow.

5.5 Test equipment

We ran our tests on two systems with the following software and hardware characteristics:

- Ubuntu 12.04.2, 64-bit kernel 3.2.0, Intel Core Dual-E2200 @ 2.20GHz, gcc 4.6.3-O3
- OS X 10.7.5, Darwin 11.4.0, Intel Core i5-2557M @ 1.8 GHz, gcc 4.2.1-O3.

Since the relative performances of the schedulers were about the same on the two systems, we report our results only for the first system. In the next two sections we first show how we used the test environment described here to define HFS packet scheduler, and then we show how we validated its performance.
6 HFS definition

Once described the architecture in Section 3 and the test environment in Section 5, it is possible to show an easy procedure to define new flexible and efficient schedulers for wireless links. Inside this environment it is easy to cherry pick from the literature and combines the best solutions in terms of service guarantees, computational cost, power consumption and throughput boosting just by combining existing high-performance schedulers for wired links. To run our tests and design new high-performance schedulers for wireless links we have considered the best schedulers for wired links available in the literature, that is:

- WF^2Q+: an optimal service guarantees with \(O(\log n)\) complexity (Bennet and Zhang, 1997)
- DRR: a scheduler with extremely low time complexity \(O(1)\), but with \(O(n)\) deviation form optimal service (Shreedhar and Varghese, 1995)
- QFQ+: a quasi-optimal service guarantees scheduler with execution time close to the DRR one (Valente, 2014).

In the following, we will describe the procedure of combining these schedulers placing them at QoS layer and/or at MAC-SAL layer and evaluating the final performance results.

6.1 QoS scheduler

From a QoS point of view, the characteristics of the schedulers are the same for both the modular architecture over wireless links and for the typical one over wired links. Due to this fact, we do not present any test results for the QoS algorithm choice and we select QFQ+ because it achieves quasi-optimal service guarantees while still preserving execution time close to the optimal, i.e., DRR (Valente, 2014). Moreover, we have verified also that the choice of the QoS scheduler does not affect any of the MAC-SAL figures of merit. Accordingly to these facts, in the next Subsection 6.2 we consider the QoS scheduler settled with QFQ+ in place.

6.2 MAC-SAL scheduler

A different evaluation must be done for the MAC-SAL layer. Here we need to consider all the metrics in field and the wireless-scenario parameters presented in Section 5.3. Inside the MAC-SAL layer, the scheduler can be configured with different weight distribution policy among flows based on their \(P_{\text{loss}}\). The distributions considered to run our experiments have been:

- Linear: considered ten flows \(f_0, \ldots, f_9\) with ten different \(P_{\text{loss}}\) ordered from the higher losses flows to the lower losses flows. With a linear weight distribution each MACSAL flow is initialised with a weight equal to \(i\), where \(i\) is the position of the flow in the ordered list.
• **Exponential**: considering the same example of the previous distribution, the weight of a generic flow in this case is initialised as $2^i$ where (again) $i$ is the position of the flow in the ordered list.

• **Analogical**: here a flow $k$ is initialised with a weight equal to $(1 - P_{loss}) \cdot 1,000$.

In this paper we show only results computed with the last distribution because it achieves the best trade off in terms of throughput boosting and QoS guarantees.

**Figure 5** Normalised throughput for different mac-sal scheduler algorithms (see online version for colours)

**Figure 6** Queueing delay introduced by different MAC-SAL scheduler algorithm (see online version for colours)
To remind that the goal of MAC-SAL scheduler is to boost the throughput, in Figure 5 it is reported the normalised throughput of different MAC-SAL schedulers: it is shown that the one achieving the best performance is QFQ+, followed very closely by WF2Q+, while DRR algorithm is the worst one. Besides, because a second scheduler placed under the QoS layer can introduce a delay, it is important to evaluate this metric also at MAC-SAL layer. In fact, in Figure 6 it is showed the queuing delay introduced by the three different schedulers to the flows/users in best channel condition: also in this case QFQ+ shows the best performance, introducing a lower delay with respect to the other schedulers. Moreover, the effective maximum delay measured is lower than the worst case delay calculated in Section 4. Finally, by looking at Figure 7, only QFQ+ reaches execution time values close to the optimal one of DRR.

Figure 7  Execution time for different MAC-SAL scheduler algorithms (see online version for colours)

6.3 HFS

After the evaluation of different schedulers placed at the QoS layer and at the MAC-SAL layer we defined HFS as the best solution, by placing QFQ+ scheduler both at the QoS layer and at the MAC-SAL layer (Casoni et al., 2014a). The cascade of two QFQ+ packet schedulers follows the flexible approach of the modular architecture: at the QoS layer QFQ+ is always the same, independently of the hardware and of the network, while at the MAC-SAL layer QFQ+ uses channel state information to improve the scheduling decision and only this last module should change moving from an hardware platform to another.

According to the Section 4, HFS scheduler gives the same service guarantees of the QoS scheduler, which is QFQ+, with an additional packet delay variation not higher than $\frac{Q'}{R'}$, whereas the additional service-lag variation is at most $Q$. 


While in this section we have considered intra-model tests to deploy and to configure the best high-performance scheduler solution compliant with the modular architecture, in the next section we will consider extra-model tests among HFS, the best schedulers for wired links and the best integrated one.

7 Results

Here we validate the efficiency of HFS packet scheduler by comparing its performance with the best packet schedulers for wired links and with the best integrated scheduler for wireless links. To validate HFS results we considered different schedulers as a benchmark:

- $W^2F^2Q$, to validate HFS throughput results against the best integrated scheduler available in the literature (Yi et al., 2000)
- $WF^2Q+$, to validate HFS guarantees against the best high-performance scheduler for wired links in terms of service guarantees
- $DRR$, to validate HFS execution time against the best high-performance scheduler for wired links in terms of time complexity.

Let us call double-SCHED a double instance of the scheduler SCHED compliant with the modular architecture with $SCHED$ placed both at QoS and at MAC-SAL layer (e.g., double-DRR is obtained placing DRR scheduler both at QoS level and at MAC-SAL level$^2$).

7.1 Throughput

The integrated scheduler $W^2F^2Q$ models temporary bursty channel errors of a wireless link only with a two-state Markov chain in order to simulate flows in good or bad channel conditions. Unfortunately, such a distinction does not hold in our model, since we considered a scenario in which a flow can experience long-term bad conditions as well, based for instance on the position of the user. Anyway, comparing $W^2F^2Q$ with HFS can benefit the first in terms of throughput, since users in good state manage the total bandwidth available leaving flows in bad state with null weight. We simulate this kind of event by using a threshold in our environment, where flows with more than 20% of packet loss are classified as flows in bad state with weight equal to zero, while the other flows are classified as flows in good state with weight equal to their own weight according to the weight distribution policy. Figure 8 shows how in this favourable case $W^2F^2Q$ scheduler obtains high normalised throughput. However, HFS achieves better throughput performance with respect to $W^2F^2Q$ for $Q \geq 100$ due to his fine-grained choice among good flows, which increases with the MAC-SAL shared buffer size $Q$. 
Achieving a high throughput and a low latency

**Figure 8** Normalised throughput for HFS and W²F²Q (see online version for colours)

7.2 **QoS guarantees**

With respect to QoS metric, we show only time worst-case fair index because it allows to evaluate, in a single graph, both fairness and latency: on one hand it shows the fairness in terms of delay from the worst-case completion time of a packet in a perfectly fair system, while on the other hand it allows to instantly calculate the actual delays incurred by packets depending on the occupation of queues. In a perfectly fair system, the worst-case completion time of a packet is equal to its queue length (including the packet itself) divided by the packet’s flow guaranteed rate. Figure 9 validates the performance of HFS in terms of QoS for flows in **intermediate conditions**, i.e., for flows/users with at least 0.8 of normalised throughput per-flow (HFS guarantees this kind of worst case performance if the station lose an amount of packets that is less or equal to 20%). With a packet loss ratio above 20% most applications do not work properly; for instance, both the TCP window and the VoIP controller mechanism do not perform well under the aforementioned condition. Why we have not considered W²F²Q in the QoS-guarantees graph? As we have said, the algorithm W²F²Q model temporary bursty channel errors of a wireless link only. This way the scheduler tries to implement the *compensation* mechanism to guarantee long term fairness between different flow-condition services under the hypothesis that an error-prone flow has sufficient time to make up for their lag after recovery of channel (see Yi et al., 2000). Unfortunately, such assumption does not hold in our model because we considered a scenario in which a flow can experience also long-term bad channel condition, based on the position of the user. In this sense, comparing W²F²Q with our model from a QoS guarantees point of view can be unfair because users/flows classified in bad state can lag forever with quasi-zero service guarantees.
7.3 Execution time

To evaluate HFS in terms of execution time (and energy consumption) we use as benchmark the scheduler double-DRR, which is the simplest possible solution to achieve high throughput preserving QoS guarantees with a low computational cost of $O(1)$ with our modular architecture. Figure 10 shows that HFS has a really close execution time to the double instance of DRR which is the best high-performance scheduler for wired links in terms of execution time. Furthermore, the picture shows that the packet-processing time of HFS is compliant with the link rate (i.e., much lower than the packet transmission time). Why we have not considered W$^2$F$^2$Q in the execution time graph? W$^2$F$^2$Q scheduling complexity is $O(N)$, too high for schedulers in backbone network where $N$ is very large and more than the HFS scheduling complexity which is close to the DRR optimal cost of $O(1)$.

Continuing the evaluation of execution time of HFS, we decided also to perform another test to stress the schedulers’ performance as a function of the number of active QoS flows. In this test HFS is configured with a MAC-SAL buffer size $Q$ equal to 100 packets. Figure 11 shows the average processing time per packet of HFS, DRR, W$^2$F$^2$Q+, W$^2$F$^2$Q and FIFO.

During this simulation the number of flows has been varied from 1 to 30,000 in order to capture the behaviour of each scheduler as a function of it. Figure 11 shows that HFS is able to maintain a $O(1)$ execution time close to the DRR and FIFO, reported as a benchmark. At the same time, these results highlight the logarithmic complexity of W$^2$F$^2$Q+ and the linear complexity of W$^2$F$^2$Q that, because of the logarithmic representation of the x-axis, appear linear and exponential respectively.
Achieving a high throughput and a low latency

Figure 10  Execution time of HFS against DRR and WF²Q+ (see online version for colours)

Figure 11  Execution time for different scheduler algorithms (see online version for colours)

7.4 Energy consumption

According to the models in Bartolini et al. (2011) and Sadri et al (2011), lower/higher relative execution times imply also lower/higher relative energy consumptions. It is the case of HFS, as showed in Figure 10, with a really close execution time to DRR which is the best high-performance packet scheduler for wired links in terms of execution time. Moreover, in Figure 8 we show that HFS achieves also higher throughput with respect to the best integrated packet scheduler for wireless links, which is WF²Q. In this way, by increasing the throughput, HFS increases the number of packets successfully transmitted per energy consumed (the number of retransmissions is also lower). Therefore, the reduction of the execution time and the throughput’s boosting permits to HFS to reduce the energy consumption.
By comparing Figures 8 and 9 we can say that it is possible to choose the desired tradeoff between throughput-boosting level and granularity of the service guarantees, by only setting the parameter Q. For instance, by setting a value of Q equal to 100 packets, HFS reaches a normalised throughput close to 90%, which is greater than the W2FQ one, while still preserving service guarantees close to the optimal ones of WFQ+.

8 Conclusions

In this paper we validated a modular architecture that decouples the task of providing QoS guarantees from the task of dealing with the idiosyncrasies of a wireless link. We extended a test-environment which allows to easily run, test, and analyse existing schedulers over this architecture. To complete the picture, we also defined HFS, a new flexible, efficient and accurate scheduler for providing QoS guarantees and high throughput over wireless links. To prove the effectiveness of HFS we showed, through experimental results, its high performance. HFS provides higher throughput with respect to wireless worst-case fair weighted fair queueing, the best integrated scheduler available so far. Furthermore, HFS provides low latency and accurate fairness, close to the optimal ones of worst-case weighted fair queueing. Last but not least, HFS guarantees execution times and energy consumptions that are close to those of a DRR. We also demonstrated the possibility to dynamically set, by changing the MAC-SAL buffer size Q, the desired trade-off between throughput-boosting level and granularity of service guarantees, a feature that allows HFS to be an adaptive packet scheduler able to tune its performance by following users and network requirements. Finally, we also highlighted a value for the Q parameter (equal to 100 packets) which permits network operators to expect a higher throughput than with the best integrated scheduler available so far, while being able in the same time to provide QoS guarantees that are close to the optimal ones.

References


Achieving a high throughput and a low latency


Notes

1 Source code is available in http://algogroup.unimore.it/people/paolo/agg-sched/test-env.tgz.
   and contains the script run_test.sh that we used to run the tests.